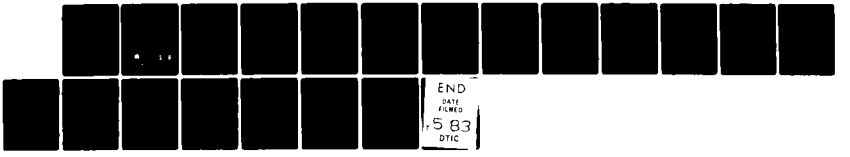
AD-A127 544 A TWO-LEVEL PIPELINED SYSTOLIC ARRAY FOR MULTI-DIMENSIONAL CONVOLUTION..(U) CARNEGIE-MELLON UNIV PITTSBURGH PA DEPT OF COMPUTER SCIENCE H T KUNG ET AL. 1/1

UNCLASSIFIED NOV 82 CMU-CS-83-103 F33615-81-K-1539 F/G 9/5 NL

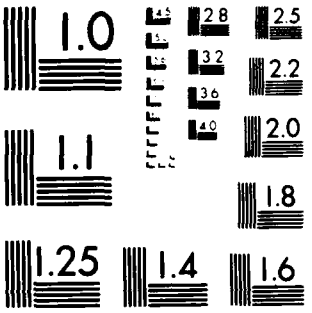


END

DATE FILMED

5 83

DTIC



MICROCOPY RESOLUTION TEST CHART
NATIONAL BUREAU OF STANDARDS 1963-A

CMU-CS-83-103

15

A Two-Level Pipelined Systolic Array for Multi-Dimensional Convolution

H. T. Kung

Lawrence M. Ruane

David W. L. Yen

November, 1982

DEPARTMENT
of
COMPUTER SCIENCE

Carnegie-Mellon University

CMU-CS-83-103

A Two-Level Pipelined Systolic Array for Multi-Dimensional Convolution

H. T. Kung
Department of Computer Science
Carnegie-Mellon University
Pittsburgh, Pennsylvania 15213
U. S. A.

Lawrence M. Ruane and David W. L. Yen
ESL Incorporated
A Subsidiary of TRW Inc.
495 Java Drive
P.O. Box 510
Sunnyvale, California 94086
U. S. A.

November, 1982

H. T. Kung was supported in part by the Defense Advanced Research Projects Agency (DoD), ARPA Order No. 3597, monitored by the Air Force Avionics Laboratory under Contract F33615-81-K-1539 and in part by the Office of Naval Research under Contracts N00014-76-C-0370, NR 044-422 and N00014-80-C-0236, NR 048-659. David W. L. Yen is currently with IBM San Jose Research Laboratory, San Jose, California 95193.

ABSTRACT

This paper describes a systolic array for the computation of n-dimensional (n-D) convolutions for any positive integer n. Systolic systems usually achieve high performance by allowing computations to be pipelined over a large array of processing elements. To achieve even higher performance, the systolic array of this paper utilizes a *second* level of pipelining by allowing the processing elements themselves to be pipelined to an arbitrary degree. Moreover, it is shown that as far as orders of magnitude are concerned, the total amount of memory required by the systolic array is no more than that needed by any convolution device that reads in each input data item only once. Thus if only schemes that use the minimum-possible I/O are considered, the systolic array is not only high performance, but also optimal in terms of the amount of required memory.

Key Words and Phrases

Multi-dimensional convolution, signal and image processing, systolic array, pipelined arithmetic unit, special-purpose processor, minimum I/O requirement, VLSI.

1. Introduction

Multi-dimensional convolutions constitute some of the most compute-intensive tasks in signal and image processing. For example, a 2-D convolution using a general 4×4 kernel requires 16 multiplications and 15 additions to generate each pixel in the output image. If the dimensionality is higher or the kernel is larger, even more arithmetic operations would be required. Though computationally demanding, convolution is nevertheless a highly regular computation. By exploiting the regularity inherent to the computation, cost-effective, high performance systolic arrays, that are capable of using many systolic cells in parallel, have been proposed or built to perform multi-dimensional convolutions [1, 2, 3, 4, 5, 8].

Previously proposed systems, however, suffer from two drawbacks. Firstly, they do not take advantages of the possibility that arithmetic units could themselves be pipelined. Secondly, they cannot be used to perform convolutions of arbitrary dimensionality—they can for example perform only 1-D or 2-D convolutions, but not both.

Section 2 of this paper describes a systolic array for 1-D convolution which utilizes pipelined arithmetic units. The systolic array is extended and optimized in Section 3 to handle multi-dimensional convolutions. By adjusting the memory size of each cell of the systolic array, the array can be used for convolutions of *any* dimensionality. It is shown in Section 4 that the total amount of memory required by the systolic array is optimal in the sense that the same amount of memory is required by any scheme that reads in each input pixel only once. The final section contains some concluding remarks.

2. 1-Dimensional Convolution

2.1. Problem Definition

Given a vector of signals $X = \{x_i\}, i = 1, 2, \ldots, n$, and another vector of weights $W = \{w_j\}, j = 1, 2, \ldots, k$, with $k \ll n$, convolving signal X with "kernel" W is to compute

$$y_s = \sum_{i=0}^{k-1} w_{i+1} \cdot x_{i+s},$$

for $s = 1, 2, \ldots, n - k + 1$. Imagine that the vector indices increase from left to right. The first result, y_1, is obtained by aligning the leftmost element of W with the leftmost element of X, then computing the inner product of W and the section of X that W overlaps. To produce the second result, y_2, the kernel slides one position rightward and the inner product calculation is again performed on the overlap. The last result, y_{n-k+1}, is obtained when rightmost element of W is aligned with the rightmost element of X.

2.2. Systolic Array for 1-D Convolution

Rather than viewing the kernel as sliding over the signal, as described above, one can consider the kernel as being fixed in space and the signal as sliding over the kernel. This view suggests a linearly-connected array for performing convolution, in which each cell holds a single kernel element throughout the computation, and the signal passes through the array, say, from left to right. An array operating in this fashion is shown in Figure 2-1. In each cell a signal element is held by a register, and the multiplier may be pipelined to an arbitrary degree. Note that moving the signal from right to left causes the kernel indices to increase right to left.

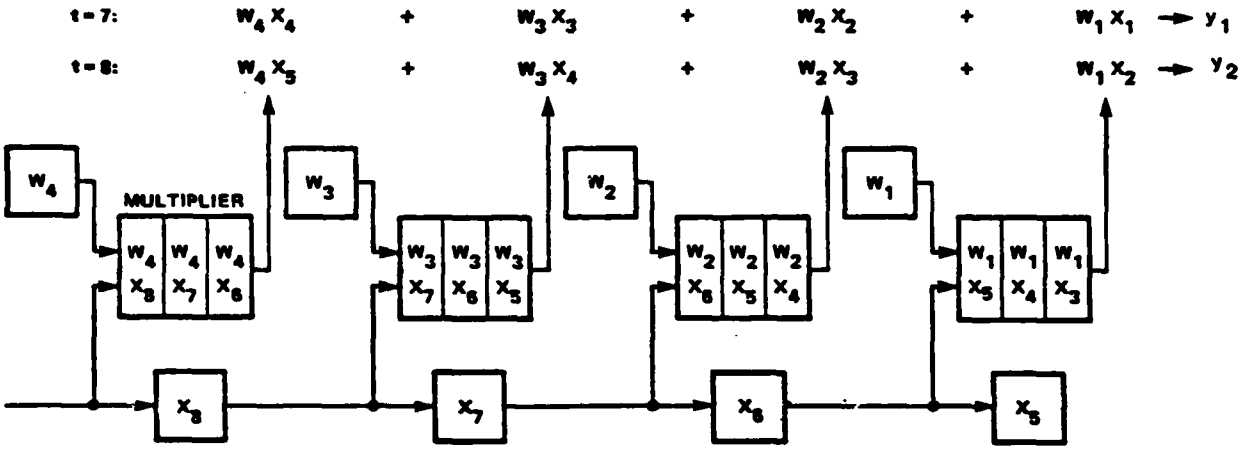


Figure 2-1: Linear-connected array for 1-D convolution.

In the figure we see that all the terms of a particular result y appear at the multiplier outputs at the same instant. They can be summed up by using a pipelined tree-adder (see, e.g., [7]). However, the interconnections needed to implement the adder tree are not local when the number of terms is large. This difficulty is avoided in the systolic approach where additions required in summing the terms of a result are apportioned to all cells uniformly and performed in different cell cycles [2, 3].

We consider the particular systolic design—design W2 in the [3]—where each cell produces its term of a result one cycle earlier than the cell to its right produces its term for the same result. The skew can be accomplished by replacing in each cell the single-stage register, which transfers the signal stream, with a two-stage shift register. For the general case when the adder in each cell is a multi-stage pipeline unit, the number of stages of the shift register should be one greater than that of the adder. The remainder of this paper assumes the systolic array structure shown in Figure 2-2 with the following parameters for its cells:

- $M =$ the number of pipeline stages of the multiplier unit,
- $A =$ the number of pipeline stages of the adder unit, and
- $R =$ the number of stages of the shift register, with $R = A + 1$.

The stage times for the multiplier, adder and shift register are assumed to be the same.

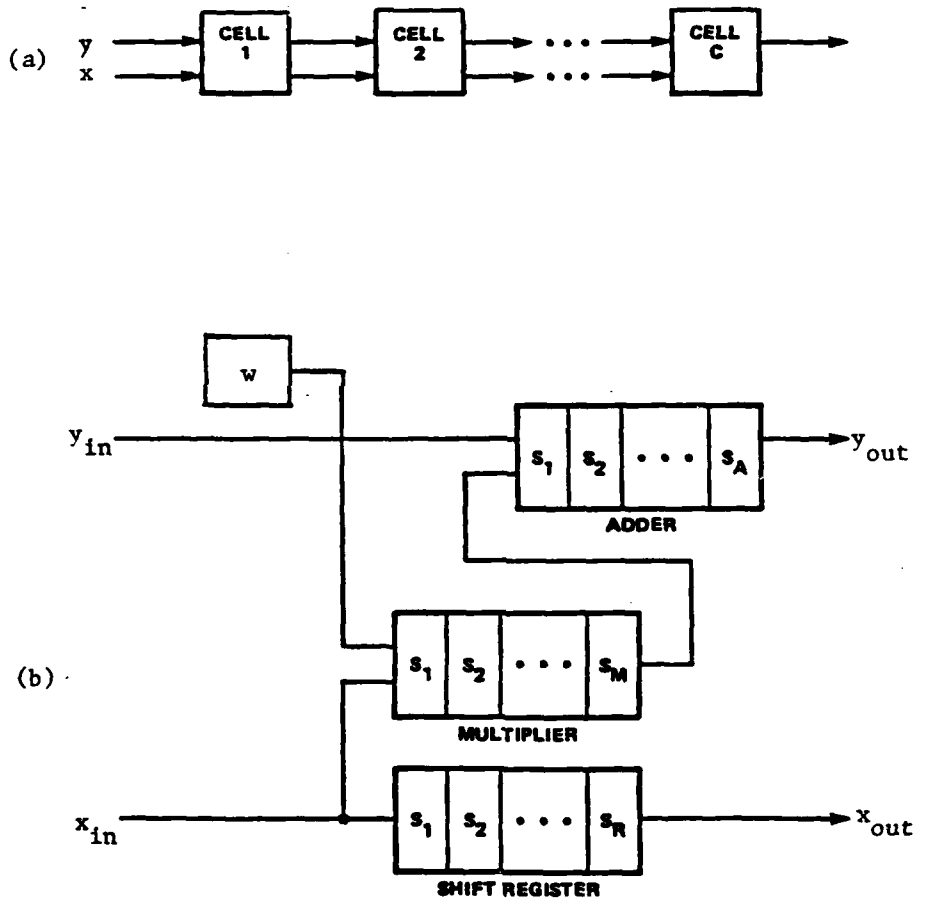


Figure 2-2: (a) Systolic 1-D convolution array and (b) its cell definition.

Figure 2-3 shows snapshots of an execution of a 1-D convolution on a 4-cell systolic array with $M = 4$, $A = 2$, and $R = 3$. In the figure, x_1 enters the array at time 0, and p_i stands for a partial result of y_i. We see that

results y_i's are output from the rightmost cell at a rate of one result per stage time of the multiplier (or adder). There is no broadcast or unbounded fan-in in the system; each cell communicates only with its immediate neighbors. This allows indefinite expansion of the array which, for electrical and timing reasons, would be impossible if global communication paths existed.

3. Multi-Dimensional Convolution

3.1. Problem Definition

Multi-dimensional convolution is a straightforward generalization of 1-D convolution, as illustrated by the definition of 2-D convolution below.

Given a 2-D signal (or image) $X = \{x_{ij}\}, i = 1,2,\ldots,m, j = 1,2,\ldots,n$, and a 2-D kernel $W = \{w_{ij}\}, i = 1,2,\ldots,k, j = 1,2,\ldots,p$, with $k \ll m$ and $p \ll n$, convolving X with W is to compute

$$y_{rs} = \sum_{i=0}^{k-1} \sum_{j=0}^{p-1} w_{i+1,j+1} \cdot x_{i+r,j+s},$$

for $r = 1,2,\ldots,m-k+1$ and $s = 1,2,\ldots,n-p+1$. Imagine that the indices increase rightward and downward. Then the first result, y_{11}, is generated by placing the kernel over the image such that w_{11} covers x_{11}, multiplying the corresponding elements of W and X, and then summing these products. The kernel slides one position to the right for the generation of the second result, y_{12}. After the last element of the first output row is computed, the kernel moves one position downward and back to the left edge of the image. As the kernel slides rightward, the second row of output is produced. Repeat this step for all rows until the last result is produced when w_{kp} covers x_{mn}.

3.2. Converting n-D Convolution into 1-D Convolution

The problem of computing an n-D convolution can be converted into that of computing a 1-D convolution, and therefore the n-D convolution can be performed on a systolic array for 1-D convolutions. For example, the 2-D convolution defined above can be viewed as a 1-D convolution with the signal and kernel defined as follows. The signal is

$$X = [x_{1*}, x_{2*}, \ldots, x_{m*}],$$

where $x_{i*} = x_{i1}, x_{i2}, \ldots, x_{in}$; that is, the signal is the concatenation of the rows of the given 2-D signal (or image) and has a total length of mn. The kernel is ($u!v$ stands for the value v repeated u times):

$$W = [w_{1*}, (n-p)!0, w_{2*}, \ldots, (n-p)!0, w_{k*}],$$

where $w_{i*} = w_{i1}, \ldots, w_{ip}$; that is, the kernel is the concatenation of the rows of the given 2-D kernel, with a

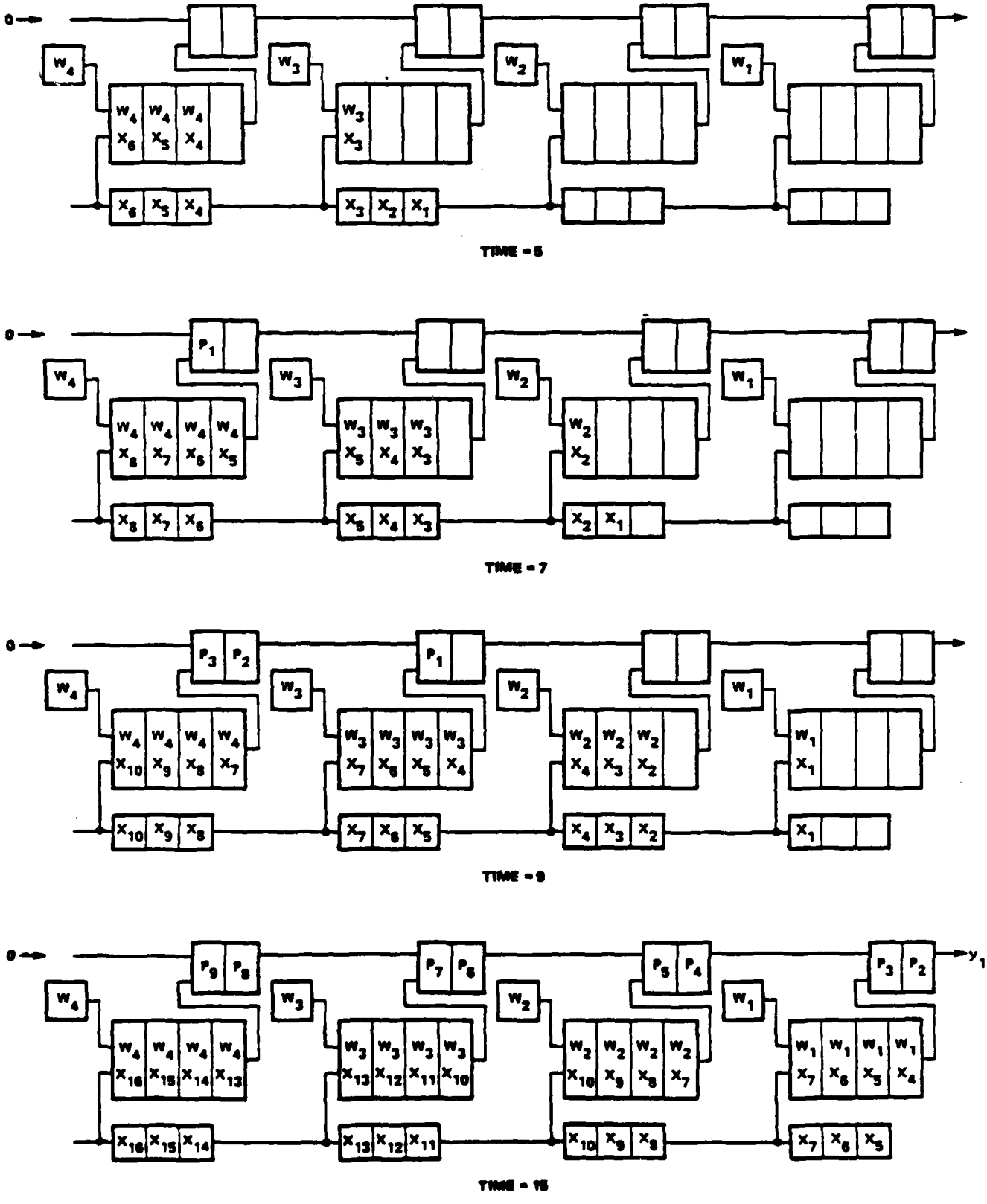


Figure 2-3: Snapshots of an execution of a systolic 1-D convolution array with $W=4$, $A=2$, and $R=3$.

vector of $(n-p)$ zero elements inserted between each consecutive pair of rows. The length of the kernel is therefore $n(k-1)+p$. To illustrate the definition, consider an example where the image is 5 rows by 4 columns ($m=5, n=4$), and the kernel is 3 rows by 2 columns ($k=3, p=2$). The 1-D signal is formed from the 2-D image as follows (an element in the image is specified by its 2-D row-column index):

$$x = [11,12,13,14,21,22,23,24,31,32,33,34,41,42,43,44,51,52,53,54],$$

and the 1-D kernel is formed from the 2-D kernel as follows:

$$W = [11,12,0,0,21,22,0,0,31,32].$$

Figure 3-1 shows snapshots of several kernel positions, each corresponding to the generation of some result, from both the 1-D and 2-D viewpoints. In the figure, kernel indices are shown on the top of signal indices. Configuration for generating the first result, y_{11}, is shown in Figure 3-1(a). The kernel moves one position to the right to produce the second result, y_{12}, as shown in Figure 3-1(b). After the last result of the first row, y_{13}, is produced, the kernel slides one position to the right. At this point an invalid result is generated, which must be ignored. This situation is shown in Figure 3-1(c). Next, the first element of the second row is produced, as shown in Figure 3-1(d). The kernel continues to slide to the right until the final result, y_{33}, is generated (Figure 3-1(e)). Although in the general case $p-1$ invalid results are generated for each row of output (except the last), the fraction of total results which are invalid is very small because $p \ll n$.

The above method of converting a 2-D problem into a 1-D problem can be generalized to that of converting an n-D problem into a 1-D problem. To illustrate the idea, consider a 3-D signal with 3 planes by 3 rows by 3 columns, and a 3-D kernel with 2 planes by 2 rows by 2 columns. The 1-D signal is formed as follows (using the 3-D plane-row-column index):

$$X = [111,112,113,121,122,123,131,132,133,211,212,213,221,222,223,231,232,233,311,312,313,$$
$$321,322,323,331,332,333],$$

and the 1-D kernel is formed as follows:

$$W = [111,112,0,121,122,0,0,0,0,211,212,0,221,222].$$

The position of the kernel for the generation of the first result is shown in Figure 3-2. The 1-D signal is formed by concatenating the rows of the first plane of the 3-D signal, followed by the rows of the second plane, etc. The 1-D kernel is formed by concatenating the rows of the first plane of the 3-D kernel, with zero vectors in between, followed by a larger vector of zeros sufficient to "cover" the remainder of the first signal plane, then followed by the rows of the second plane of the 3-D kernel with zero vectors in between, etc. One can check that a total of 14 results are generated, of which 8 are valid. As the size of the 3-D signal increases, the fraction of results which are invalid becomes very small, as in the 2-D case.

```
      11  12
      11  12  13  14

      21  22
      21  22  23  24

      31  32
2D:   31  32  33  34                          first result

      41  42  43  44

      51  52  53  54

      11 12  0  0 21 22  0  0 31 32
1D:   11 12 13 14 21 22 23 24 31 32 33 34 41 42 43 44 51 52 53 54
```

```
          11  12
      11  12  13  14

          21  22
      21  22  23  24

          31  32
2D:   31  32  33  34                          second result

      41  42  43  44

      51  52  53  54

         11 12  0  0 21 22  0  0 31 32
1D:   11 12 13 14 21 22 23 24 31 32 33 34 41 42 43 44 51 52 53 54
```

Figure 3-1: Snapshots of a 2-D convolution computation from both 1-D and 2-D viewpoints.

```
                                  11
      11  12  13                  14

      12
      21  22  23                  24

      22                          31
2D:   31  32  33                  34          invalid result
                                              (ignored)
      32
      41  42  43  44

      51  52  53  54

                    11 12  0  0 21 22  0  0 31 32
1D:   11 12 13 14 21 22 23 24 31 32 33 34 41 42 43 44 51 52 53 54
```

```
      11  12  13  14

      11  12
      21  22  23  24

      21  22
2D:   31  32  33  34                      first result of the
                                          second row
      31  32
      41  42  43  44

      51  52  53  54

                    11 12  0  0 21 22  0  0 31 32
1D:   11 12 13 14 21 22 23 24 31 32 33 34 41 42 43 44 51 52 53 54
```

```
      11  12  13  14

      21  22  23  24

                  11  12
2D:   31  32      33  34                  final result

                  21  22
      41  42      43  44

                  31  32
      51  52      53  54

                                  11 12  0  0 21 22  0  0 31 32
1D:   11 12 13 14 21 22 23 24 31 32 33 34 41 42 43 44 51 52 53 54
```

Figure 3-1 Cont.

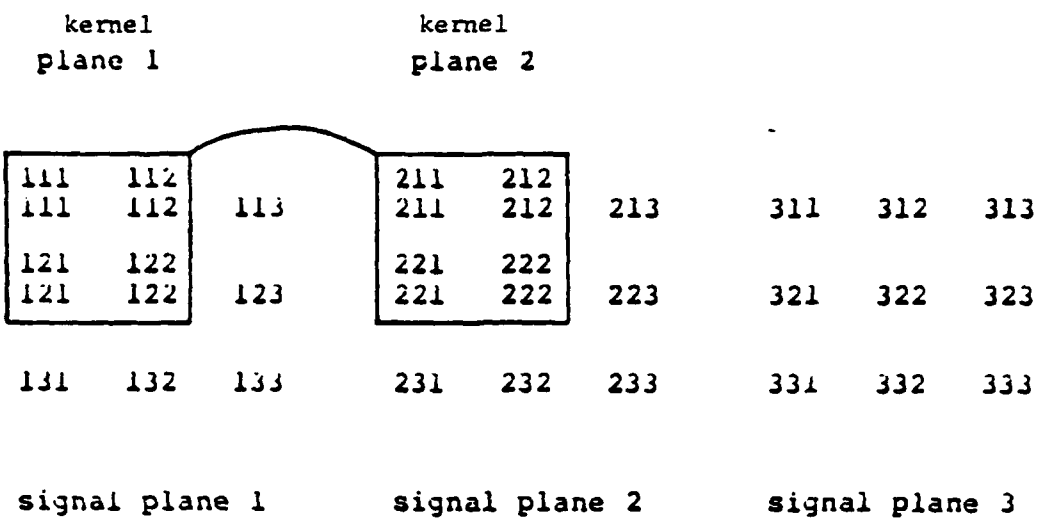


Figure 3-2: Generation of the first result of a 3-D convolution.

3.3. "Optimized" Systolic Array for Multi-Dimensional Convolution

From the preceding discussion, we see that when an n-D convolution is converted into a 1-D convolution, a kernel containing a large number of zero elements may be introduced. If the systolic array of Section 2.2 is applied to perform the 1-D convolution directly, then a large number of cells would be needed in the array, and cells with zero weights would perform no useful work. We show here that a kernel containing a large number of zeros can be accommodated on a small array, for which the number of stages of the shift register in each individual cells is adjustable.

Consider a cell containing a zero kernel element. From Figure 2-2, we see that the only effects of that cell are to delay the y stream by A cycles and the x stream by $R = A + 1$ cycles. Therefore if this cell is replaced with a cell that just introduces zero cycle delay for the y stream (a direct path) and a single cycle delay for the x stream, the same result stream would be generated, although each result would appear A cycles earlier than before. This degenerate cell may in turn be absorbed into the cell to the left by increasing the number of shift register stages of that cell by one, to $A + 2$. From these observations, we note that the systolic array needs only q cells, where q is the number of non-zero elements in the kernel; these non-zero elements are loaded into consecutive cells of the array. For $i = 1, 2, \ldots, q-1$, let Z_i be the number of zero kernel elements between the kernel elements stored in cell i and cell $i+1$, and R_i the number of stages of the shift register in cell i. The shift registers are configured such that

$$R_i = A + 1 + Z_i.$$

As an example, if $A = 2$ and $W = [3,0,0,5,9,0,6]$, only 4 cells are needed, with weight 6 stored in cell 1, 9 in cell 2, 5 in cell 3, and 3 in cell 4, and with $[R_1,R_2,R_3] = [4,3,5]$. A series of snapshots of a 2-D convolution computation on this systolic array are show in Figure 3-3.

If $A + 1 + Z_i$ exceeds the physical capacity of the shift register of cell i, the problem can still be solved by using dummy cells. For example a dummy cell i', having a zero as its kernel element, can be introduced between cell i and cell $i + 1$, such that $R_i + R_{i'} = 2A + 1 + Z_i$.

The "cell-saving" technique described in this section is useful for 1-D convolutions derived from n-D convolutions, because in this case 1-D kernels contain large numbers of zero elements. The following theorem summarizes the properties of the systolic array that is capable of solving the 2-D convolution problem considered in Sections 3.1 and 3.2.

> **Theorem 1:** The convolution of an $m \times n$ image with a $k \times p$ kernel ($k \le p$) can be performed on a linearly-connected systolic array of kp cells, where cells ip, $i = 1, 2, \ldots, k-1$, each have a shift register of $A + 1 + n - p$ stages, and the rest of the cells each have a shift register of $A + 1$ stages. (A total of $O(kn)$ memory is thus needed for all the shift registers.)

A theorem corresponding to the case when $p < k$ can be obtained in a similar way.

4. Memory Requirement

The systolic convolution array described above relies on the availability long shift registers in many cells. Is this an excessive use of memory? This section shows that as far as orders of magnitude are concerned, the total amount of memory required by the systolic array is no more than that needed by any convolution device that reads in each input pixel only once. Thus if we restrict ourselves to schemes that use the minimum-possible I/O, the systolic array is not only high performance, but also optimal in terms of the amount of required memory.

To be precise, we model in Figure 4-1 the I/O behavier of a special-purpose convolution device. The convolution device retrieves input pixels from some external memory and performs computations on them. Computed results are stored back either to the same external memory as shown in the figure or to another memory. The external memory is typically a disk, rather than the fast memory of a computer, in order to store large images. In this case, I/O with the external memory is relatively slow; we assume that it is necessary that each input pixel originally stored in the external memory be brought out to the convolution device only once so that I/O will not be a bottleneck. To accomplish this, some form of memory assigned specifically to the convolution device, called the device memory here, is needed, since each input pixel is used multiple times to compute multiple output pixels. The device memory may be a line buffer capable of holding a

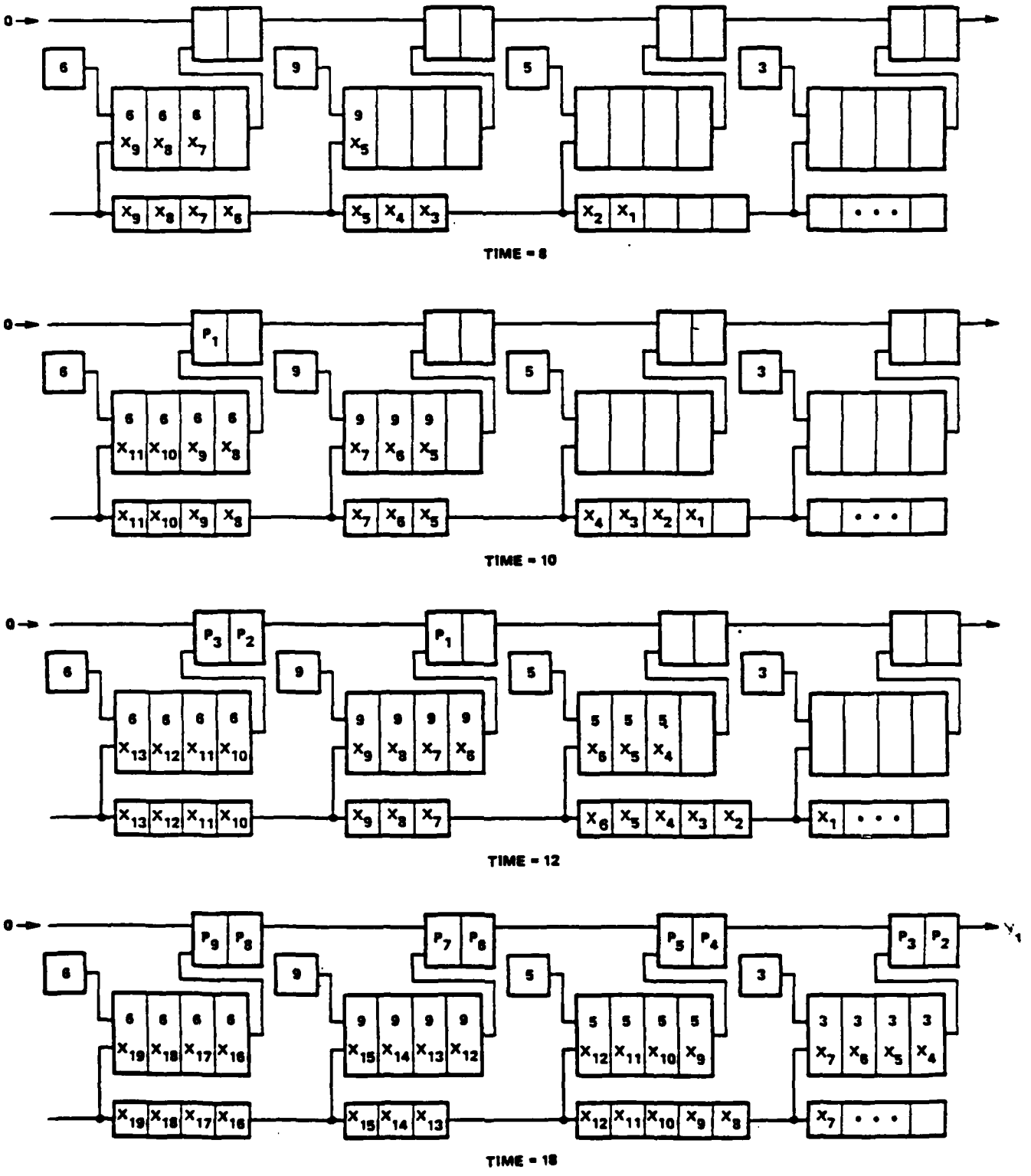


Figure 3-3: Snapshots of a 2-D convolution computation on a linearly-connected systolic array.

number of image lines (see, e.g., [4]), or a memory distributed over cells of the device as in the case of the systolic convolution array described in the preceding section. Theorem 3 below establishes a lower bound on the size of the device memory, i.e., the number of pixel values it must hold, for any 2-D convolution device. The proof can be generalized to the n-D case for any n in a straightforward way.

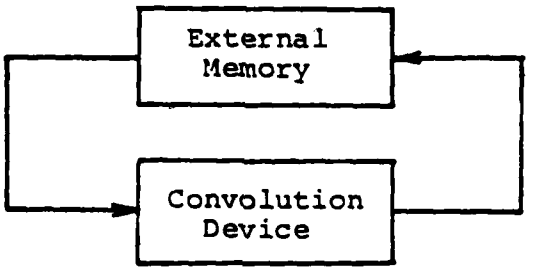


Figure 4-1: I/O model of a convolution device.

We make an important (and quite reasonable) assumption that the convolution device treats elements in the image and in the kernel as indeterminates. Thus the device never looks into values of the elements, nor possible relations among them, in order to decide what to do next. In other words, the design or the program of the device is data independent.

The proof of the theorem is based on an information-theoretic argument and uses the the following well known result [6]:

> **Lemma 2:** To bisect an $m \times n$ mesh-connected graph at least $c \cdot n$ edges have to be removed, where $c > 0$ is a constant independent of n.

Bisecting a graph means partitioning the graph into two subgraphs, each containing about half of the nodes of the original graph. Here we assume that the number of nodes in each subgraph is within the range of $(1/2)mn \pm 2kn$.

> **Theorem 3:** Consider convolving an $m \times n$ image with a $k \times p$ kernel on a convolution device, where $n \leq m$, $k \leq p$, and m, n are arbitrarily large. If the device is allowed to read in each input pixel only once, then it requires a memory of size at least proportional to kn.
>
> **Proof:** Assume that there is a moment when the number of pixels that has been output by the convolution device is within the range $(1/2)mn \pm kn$. (It will be shown at the end of the proof that this assumption is without loss of generality.) At this moment, let U be the set of pixel locations in the input image whose values have been input from the external memory to the device memory, and V the set of pixel locations in the output image whose values have been output from the convolution device. Consider the $m \times n$ mesh-connected graph. Notice that V and its complement define a bisection of the graph. For $i = 1, 2, \ldots, k-1$, define $V^{(i)}$ to be the union of V and the set

of those nodes which are in the complement of V and reachable from some node in V by a path of length no more than i. Then by Lemma 2 we have

$$|V^{(1)}| - |V| \geq cn,$$

and

$$|V^{(i+1)}| - |V^{(i)}| \geq cn,$$

for $i = 1, 2, \ldots, k-2$, where $|V|$ and $|V^{(i)}|$ are the cardinalities of V and $V^{(i)}$, respectively. Since by the definition of the convolution problem $V^{(k-1)} \subseteq U$, we have

$$|U| - |V| \geq (k-1)cn.$$

Consider now the input pixels whose pixel locations are in $U - V$. Note that these input pixels are needed to compute output pixels at corresponding locations, and that they cannot be regenerated from future inputs from the external memory. Thus they (or some equivalent amount of information to specify them) must be stored in the device memory. Since the input pixels are assumed to be indeterminates, a memory of size at least $|U| - |V|$, or $(k-1)cn$, is needed to specify these pixel values at this moment.

Suppose that the convolution device generates output pixels in large bursts so that no such moment exists when the number of pixels that have been output is within the range $(1/2)mn \pm kn$. Then there must be a moment when a set of at least $2kn$ output pixels are generated in a single burst. For producing these output pixels in a single burst all the input pixels at the corresponding pixel locations (or equivalent amount of information) must be stored in the device memory immediately before the burst.

By Theorem 3 we conclude that the $O(kn)$ memory needed by the systolic array of Theorem 1 for the 2-D convolution is the minimum-possible, if we assume that each input pixel is allowed to be read in from the external memory only once.

5. Concluding Remarks

The systolic array of this paper can execute n-D convolutions with only as many cells as there are non-zero elements of the kernel. Note that zeros in the kernel arise not only from conversion of n-D problems to 1-D problems, but also directly from applications. The shift registers in the cells can probably be most economically implemented using RAM chips, since the size and cost of these chips are decreasing at a very rapid rate, and are likely to continue to do so.

The cells of the array may be considered as the individual segments of a pipeline. In addition, the adder and multiplier of each cell may be pipelined to any degree. This two-level pipelining provides high computa-

tion rates and is suitable for complex arithmetic units, such as floating point units, which are typically implemented as multi-stage pipelines.

The approach presented for multi-dimensional convolutions has the advantage that the signal is accessed in "natural" row (or column) order, which corresponds to the way the signal is stored in the linear memory of the host. This allows the use of direct memory access (DMA) rather than requiring addresses to be generated with software, which should enhance throughput significantly. The same remarks apply to the generation of the results.

Finally, if the number of cells in the systolic array is insufficient for a particular problem, the problem can be decomposed by passing the signal through the array multiple times, each time with a successive segment of the kernel. For the second and successive passes, the results of the previous pass are entered into the adder of the leftmost cell (the *y* input), rather than a constant zero stream. This assumes that the signal can be accessed several times (i.e., the processor is not being used in a real time, flow-through mode), and that a path exists between the output and the *y* input of the array.

References

[1] Blackmer, J., P. Kuekes and Frank, G.
A 200 MOPS Systolic Processor.
In *Proceedings of SPIE Symposium, Vol. 298, Real-Time Signal Processing IV*. The Society of Photo-optical Instrumentation Engineers, August, 1981.

[2] Kung, H.T.
Special-Purpose Devices for Signal and Image Processing: An Opportunity in VLSI.
In *Proceedings of the SPIE, Vol. 241, Real-Time Signal Processing III*, pages 76-84. The Society of Photo-Optical Instrumentation Engineers, July, 1980.

[3] Kung, H.T.
Why Systolic Architectures?
Computer Magazine 15(1):37-46, January, 1982.

[4] Kung, H.T. and Picard, R.L.
Hardware Pipelines for Multi-Dimensional Convolution and Resampling.
In *Proceedings of the 1981 IEEE Computer Society Workshop on Computer Architecture for Pattern Analysis and Image Database Management*, pages 273-278. IEEE Computer Society Press, November, 1981.

[5] Kung, H.T. and Song, S.W.
A Systolic 2-D Convolution Chip.
In Preston, K., Jr. and Uhr, L. (editor), *Multicomputers and Image Processing: Algorithms and Programs*, pages 373-384. Academic Press, 1982.
An extended abstract appears in *Proceedings of 1981 IEEE Computer Society Workshop on Computer Architecture for Pattern Analysis and Image Database Management*, November 11-13, 1981, pp. 159-160.

[6] Lipton, R.J., Eisenstat, S.C. and DeMillo, R.A.
Space and Time Hierarchies for Classes of Control Structures and Data Structures.
Journal of the ACM 23(4):720-732, October, 1976.

[7] Swartzlander, E.E., Jr. and Gilbert, B.K.
Arithmetic for Ultra-High-Speed Tomography.
IEEE Transactions on Computers C-29(5):341-354, May, 1980.

[8] Yen, D.W.L. and Kulkarni, A.V.
Systolic Processing and an Implementation for Signal and Image Processing.
IEEE Transactions on Computers C-31(10):1000-1009, October, 1982.

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

REPORT DOCUMENTATION PAGE	READ INSTRUCTIONS BEFORE COMPLETING FORM
1. REPORT NUMBER: CMU-CS-83-103	2. GOVT ACCESSION NO. AD-A127 542
3. RECIPIENT'S CATALOG NUMBER	
4. TITLE (and Subtitle): A TWO-LEVEL PIPELINED SYSTOLIC ARRAY FOR MULTI-DIMENSIONAL CONVOLUTION	5. TYPE OF REPORT & PERIOD COVERED: Interim
	6. PERFORMING ORG. REPORT NUMBER
7. AUTHOR(s): H.T.Kung L.M. Ruane, David W.L.Yen from ESL, INC. a Subsidiary of TRW, INC.	8. CONTRACT OR GRANT NUMBER(s): F33615-81-K-1539, NR04442 N00014-80-C-0236, NR04865 N00014-76-C-0370
9. PERFORMING ORGANIZATION NAME AND ADDRESS: Carnegie-Mellon University Computer Science Department Pittsburgh, PA. 15213	10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS
11. CONTROLLING OFFICE NAME AND ADDRESS: Office of Naval Research Arlington, VA 22217	12. REPORT DATE: November 1982 13. NUMBER OF PAGES: 18
14. MONITORING AGENCY NAME & ADDRESS (if different from Controlling Office)	15. SECURITY CLASS. (of this report): UNCLASSIFIED 15a. DECLASSIFICATION/DOWNGRADING SCHEDULE

16. DISTRIBUTION STATEMENT (of this Report)

Approved for Public release; Distribution unlimited

17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)

Approved for public release; distribution unlimited

18. SUPPLEMENTARY NOTES

19. KEY WORDS (Continue on reverse side if necessary and identify by block number)

20. ABSTRACT (Continue on reverse side if necessary and identify by block number)

DD FORM 1 JAN 73 1473 EDITION OF 1 NOV 65 IS OBSOLETE
S/N 0102-014-6601

UNCLASSIFIED
SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)